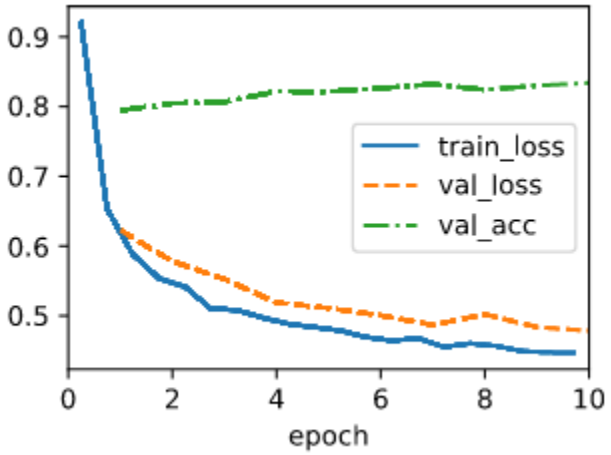
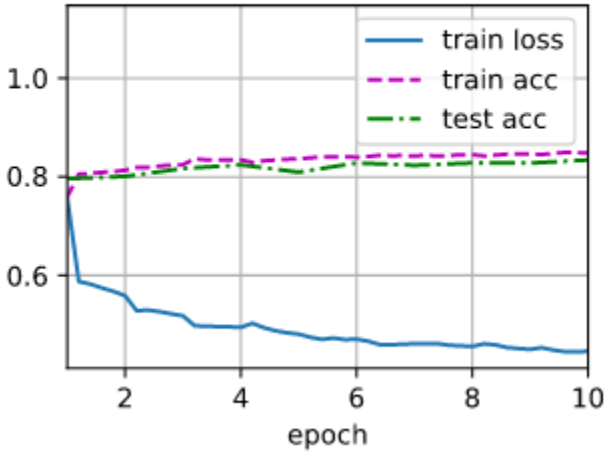
ECGR-5106: Homework 2
Patrick Flynn | 801055057
2023/2/20
Github: https://github.com/pflynn157/ecgr-5106

Problem 1, Part 1

The first part of problem 1 simply verifies that everything works properly. Here, we build the FashionMNIST dataset and run it on a simple SoftmaxRegression model. We also set up the training function so we can show the models later on. Specifically, throughout this homework, we show two graphs: one with training loss, validation loss, and validation accuracy, and another with training loss, training accuracy, and test accuracy. Here are two images from this initial test to give a sense of what it will look like in the remaining problems:




```
loss 0.446, train acc 0.848, test acc 0.834
72375.9 examples/sec on cuda:0
```



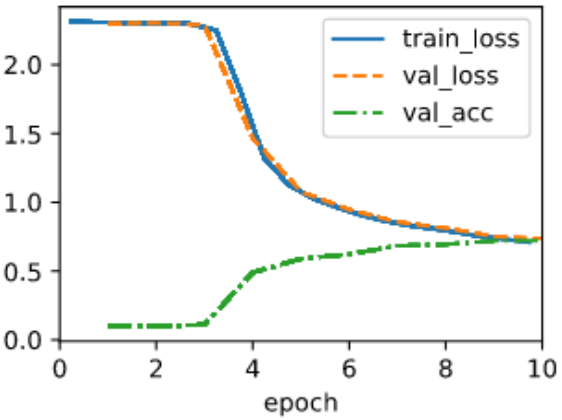
Problem 1, Part 2

Next, we implement a convolutional neural network, specifically LeNet. This follows the standard set up of a convolutional network with sigmoid backpropagation, and alternating layers of convolutional networks followed by average pooling, flattening, and a lazy-linear network.

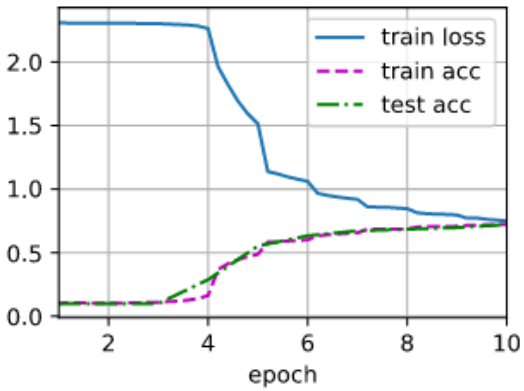
The model has this shape:

```
Conv2d output shape:     torch.Size([1, 6, 28, 28])
Sigmoid output shape:    torch.Size([1, 6, 28, 28])
AvgPool2d output shape:          torch.Size([1, 6, 14, 14])
Conv2d output shape:     torch.Size([1, 16, 10, 10])
Sigmoid output shape:    torch.Size([1, 16, 10, 10])
AvgPool2d output shape:          torch.Size([1, 16, 5, 5])
Flatten output shape:    torch.Size([1, 400])
Linear output shape:     torch.Size([1, 120])
Sigmoid output shape:    torch.Size([1, 120])
Linear output shape:     torch.Size([1, 84])
Sigmoid output shape:    torch.Size([1, 84])
Linear output shape:     torch.Size([1, 10])
```

And yields the following training data:




```
loss 0.750, train acc 0.722, test acc 0.722
26734.5 examples/sec on cuda:0
```



This network concludes the initial setting up and verification of everything. The following networks implement the actual homework assignments.

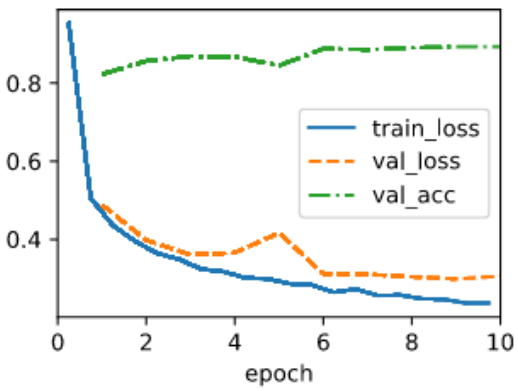
Problem 1: Implementation

In this problem, we replaced average pooling with max pooling and the softmax layer with the sigmoid layer from the previous, initial LeNet implementation. Doing this yielded the following results:

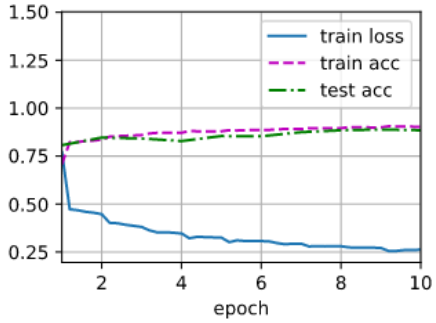
The shape:

```
Conv2d output shape:     torch.Size([1, 6, 28, 28])
ReLU output shape:       torch.Size([1, 6, 28, 28])
MaxPool2d output shape:          torch.Size([1, 6, 14, 14])
Conv2d output shape:     torch.Size([1, 16, 10, 10])
ReLU output shape:       torch.Size([1, 16, 10, 10])
MaxPool2d output shape:          torch.Size([1, 16, 5, 5])
Flatten output shape:    torch.Size([1, 400])
Linear output shape:     torch.Size([1, 120])
ReLU output shape:       torch.Size([1, 120])
Linear output shape:     torch.Size([1, 84])
ReLU output shape:       torch.Size([1, 84])
Linear output shape:     torch.Size([1, 10])
```

The actual results:



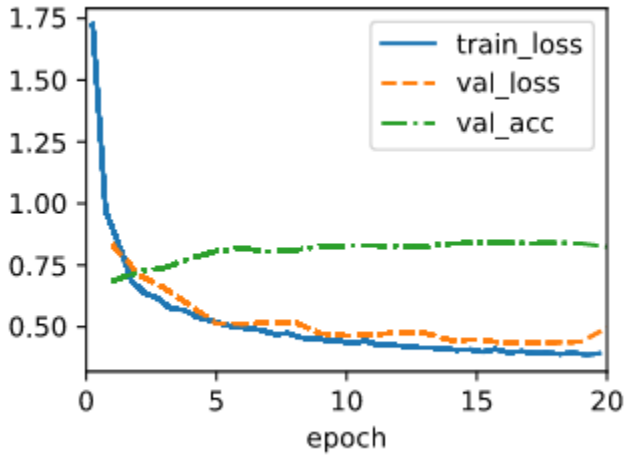

```
loss 0.261, train acc 0.903, test acc 0.885
26988.8 examples/sec on cuda:0
```



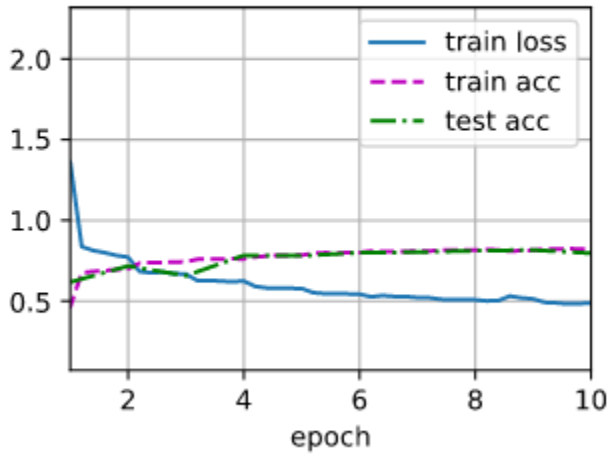
Problem 2:

In this problem, we attempted to optimize the LeNet style of neural networks by making various changes to the convolution window sizes, output channels, layers, and so on. Various modifications were tried, but in the end, increasing the number of layers and making various adjustments to the kernel and channel sizes seemed to yield the greatest effects.

Here are the results from training. Note that while some dimensions may appear smaller, the dimensions to the graph have increased, yielding this effect:



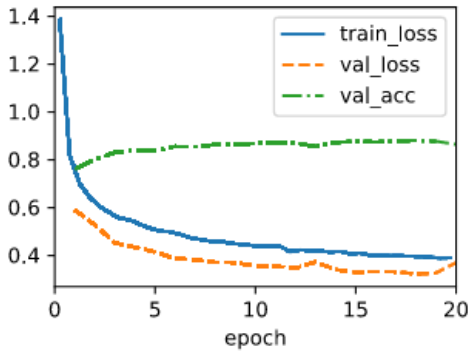

```
loss 0.486, train acc 0.823, test acc 0.796
18697.0 examples/sec on cuda:0
```



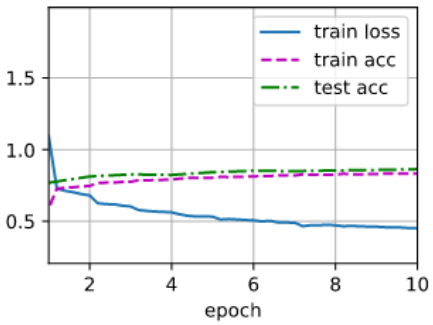
Problem 3:

In problem 3, we applied dropout to the LeNet five model to see the results. We tested this on two models: the original, unmodified LeNet model, and the modified model from the previous problem. In summary, applying dropout to the original LeNet model didn't yield huge changes; the results were similar, though the training and test accuracy did improve, while the validation accuracy smoothed some.

Here are the results:

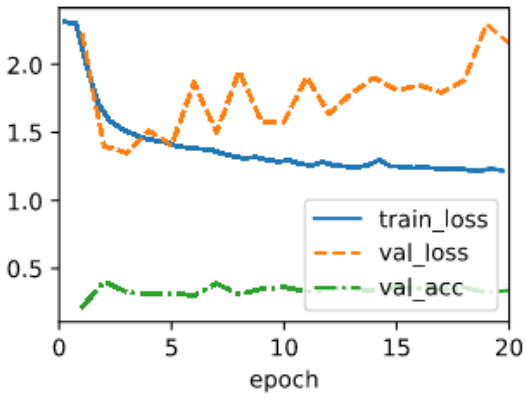



```
loss 0.452, train acc 0.832, test acc 0.864
25672.5 examples/sec on cuda:0
```

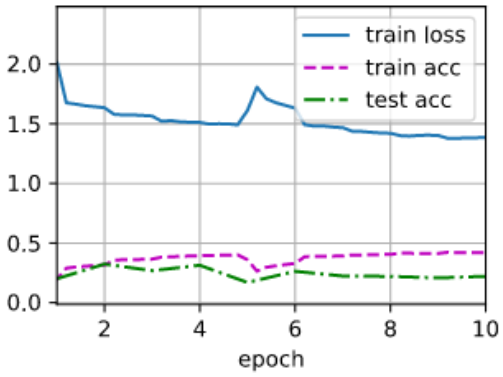


Applying dropout to the modified model did not yield good results. The training and validation loss sharply increased. In this case, applying dropout to the modified LeNet model is probably not the best solution.

Here are the results:

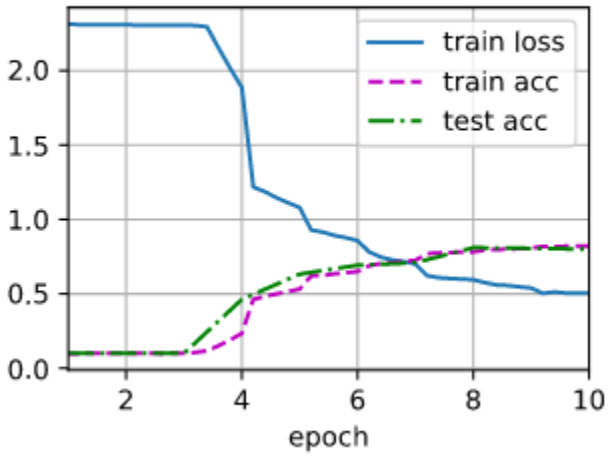



```
loss 1.384, train acc 0.420, test acc 0.222
17338.4 examples/sec on cuda:0
```



Problem 4:
In problem 4, we applied AlexNet to FashionMNIST. Some modifications to this model were necessary since it is primarily meant for bigger models than FashionMNIST. Nevertheless, with some modifications, we were able to get this to work. The results were not entirely unsimilar from those of running the dataset on an unmodified LeNet network.
Here are the results:

```
loss 0.503, train acc 0.822, test acc 0.800
9714.4 examples/sec on cuda:0
```



Note to Grader:
I am aware of the requirement to test things with the ptflops library. However, I spent a good part of an afternoon trying to get it to work, and it kept returning an error that I couldn't figure out, despite google searches and a rough idea language-wise of what it meant. I also asked friends for help, and even on comparing my code to theirs, we didn't see anything obvious that could be causing this. Due to work and personal things going on last week, I ran out of time to ask for more help and investigate more deeply.